UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

Tecnologias e Programação de Sistemas de Informação

Exceções e SWING em JAVA

Arquitetura de Dispositivos | David Jardim

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

Da aula anterior…

- Enumerados e Coleções

Mecanismo de excepções do Java

- Usado para
  - Excepções propriamente ditas (casos excepcionais)
  - Erros de programação
  - Erros irrecuperáveis

- Excepções e erros:
  - São *lançados* (a ver mais tarde)
  - Podem ser *apanhados*
  - Organizados numa *hierarquia* (como uma taxonomia)

Mas erros irrecuperáveis não devem ser apanhados!

Representação e criação

- Em Java, casos excepcionais, erros de programação e erros no ambiente de execução representam-se através de *lançáveis* (designam-se em geral por *excepções*)

- As excepções são criadas e lançadas
  - Explicitamente: throw
  - Implicitamente: assert
  - Automaticamente: JVM, por exemplo
  - Por métodos sobre os quais não temos controlo

Detecção e tratamento

- As excepções em Java podem ser capturadas
  - Para lidar integralmente com um caso excepcional
  - Para lidar parcialmente com um caso excepcional
  - Apenas para “arrumar a casa” , quando não se sabe lidar com um caso excepcional
  - Para terminar o programa de forma adequada em caso de erro de programação ou de erro no ambiente de execução

Isto não é uma recuperação, mas sim uma finalização elegante

Hierarquia de erros e excepções em Java

- Throwable
  - Error
    - VirtualMachineError
    - …
  - Exception
    - IOException
    - …
    - RuntimeException
      - ArithmeticException
      - NullPointerException
      - IndexOutOfBoundsException
      - …

Erros irrecuperáveis, que não é razoável apanhar e processar.

Excepções propriamente ditas (casos excepcionais) cujo possível lançamento é *obrigatório declarar e capturar ou delegar*.

Erros de programação, cujo possível lançamento *não se declara*.

Lançamento: caso excepcional

- Sempre que ocorre situação excepcional…
  …lançar uma excepção de uma subclasse de `Exception`
- Exemplo abrir um ficheiro:

```
public … open(…) throws FileNotFoundException {
    …
    if (file == null)
        throw new FileNotFoundException();
    …
}
```

Lançamento: erro de programação

- Sempre que se detecta um erro de programação…
  …lançar uma excepção de uma subclasse de `RuntimeException`
- Exemplo calcular a raiz quadrada:

```
public double sqrt(final double value) {
    if (value < 0.0)
        throw new IllegalArgumentException();
    …
}
```

Lidando com caso excepcional

```
public void someMethod(…) {
    try {
        …
        anObject.someOtherMethod(…);

        … // only if no exception is thrown.
    } catch (final SomeException exception) {
        … // fix the problem using information available.
    }

    … // continue execution.
}
```

Lidando com caso excepcional e o bloco finally

```
public void someMethod(…) {
    try {
        …
        anObject.someOtherMethod(…);
        …
    } catch (final SomeException exception) {
        …
    } finally {
        … // cleanup code in any case.
       anObject.close();
    }
    …
}
```

Bloco `finally` executado mesmo no caso de excepções não capturadas, retornos, novos lançamentos, asserções falhadas, etc.

Usado para colocar código de "limpeza", fechar streams por exemplo.

Excepções definidas pelo programador

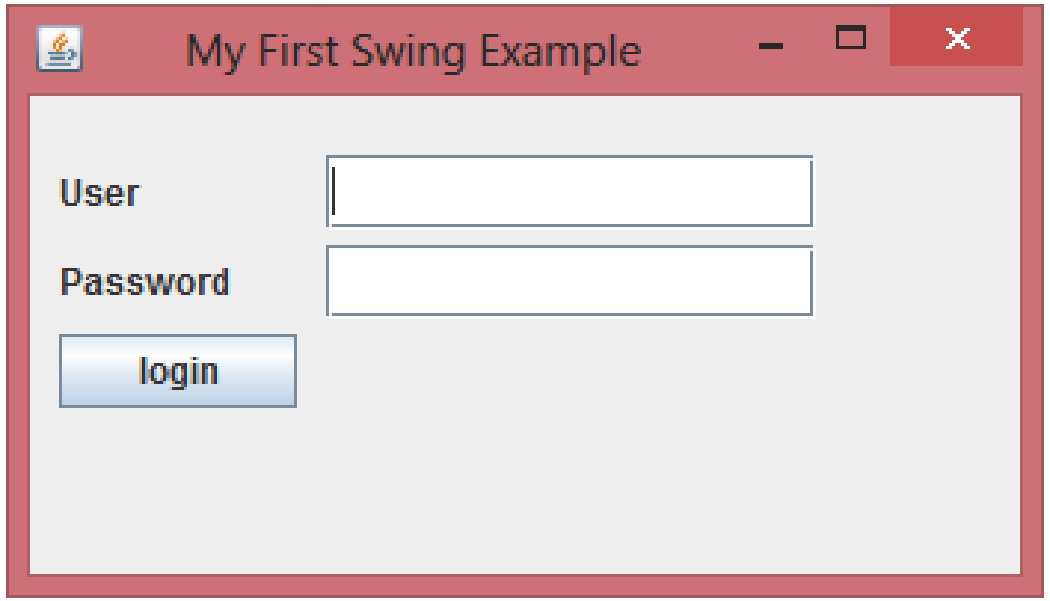
```
public class SomeException extends Exception {

    public Exception() {
    }
    public Exception(final String message) {
        super(message);
    }

    public Exception(final String message,
                     final Throwable cause) {
        super(message, cause);
    }

    public Exception(final Throwable cause) {
        super(cause);
    }
}
```

Exemplos RuntimeException

Excepção	Utilização
`IllegalArgumentException`	Argumento de método não é válido.
`IllegalStateException`	Estado de objecto é inválido ou não permite a operação.
`NullPointerException`	Tentativa de acesso a instância através de referência nula.
`IndexOutOfBoundsException`	Índice de matriz ou colecção fora dos limites válidos.
`ConcurrentModificationException`	Tentativa de alteração de objecto em momento em que tal não é permitido (e.g., alteração de lista concorrente com iteração dessa lista).
`UnsuportedOperationException`	Classe do objecto não suporta a operação em questão.

JAVA SWING

- O Java Swing providencia um conjunto de bibliotecas para criar interfaces gráficas (Graphical User Interface - GUI)



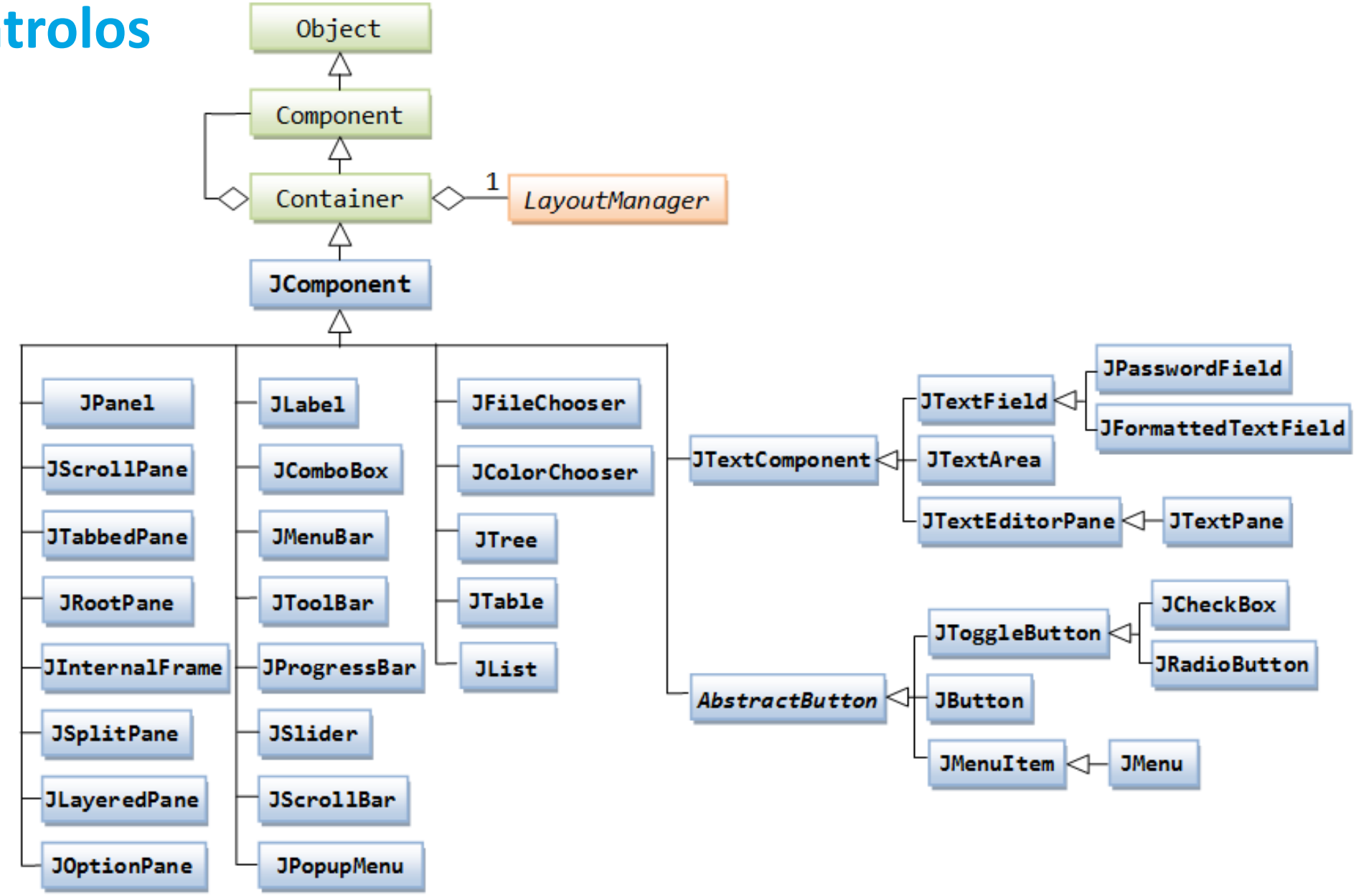
Controlos

- **Elementos da UI** - esses são os elementos visuais que o utilizador vê e interage. O GWT possui uma lista de elementos (botões, labels, caixas de texto, etc)

- **Layouts** - Definem como os elementos da interface devem ser organizados e fornecem uma aparência final para a GUI (Graphical User Interface

- **Comportamento** - Estes são os eventos que ocorrem quando o utilizador interage com elementos da interface

Controlos


Object
Component
Container
1
LayoutManager
JComponent
JPanel
JScrollPane
JTabbedPane
JRootPane
JInternalFrame
JSplitPane
JLayeredPane
JOptionPane
JLabel
JComboBox
JMenuBar
JToolBar
JProgressBar
JSlider
JScrollBar
JPopupMenu
JFileChooser
JColorChooser
JTree
JTable
JList
JTextComponent
JTextField
JPasswordField
JFormattedTextField
JTextArea
JTextEditorPane
JTextPane
AbstractButton
JToggleButton
JCheckBox
JRadioButton
JButton
JMenuItem
JMenu

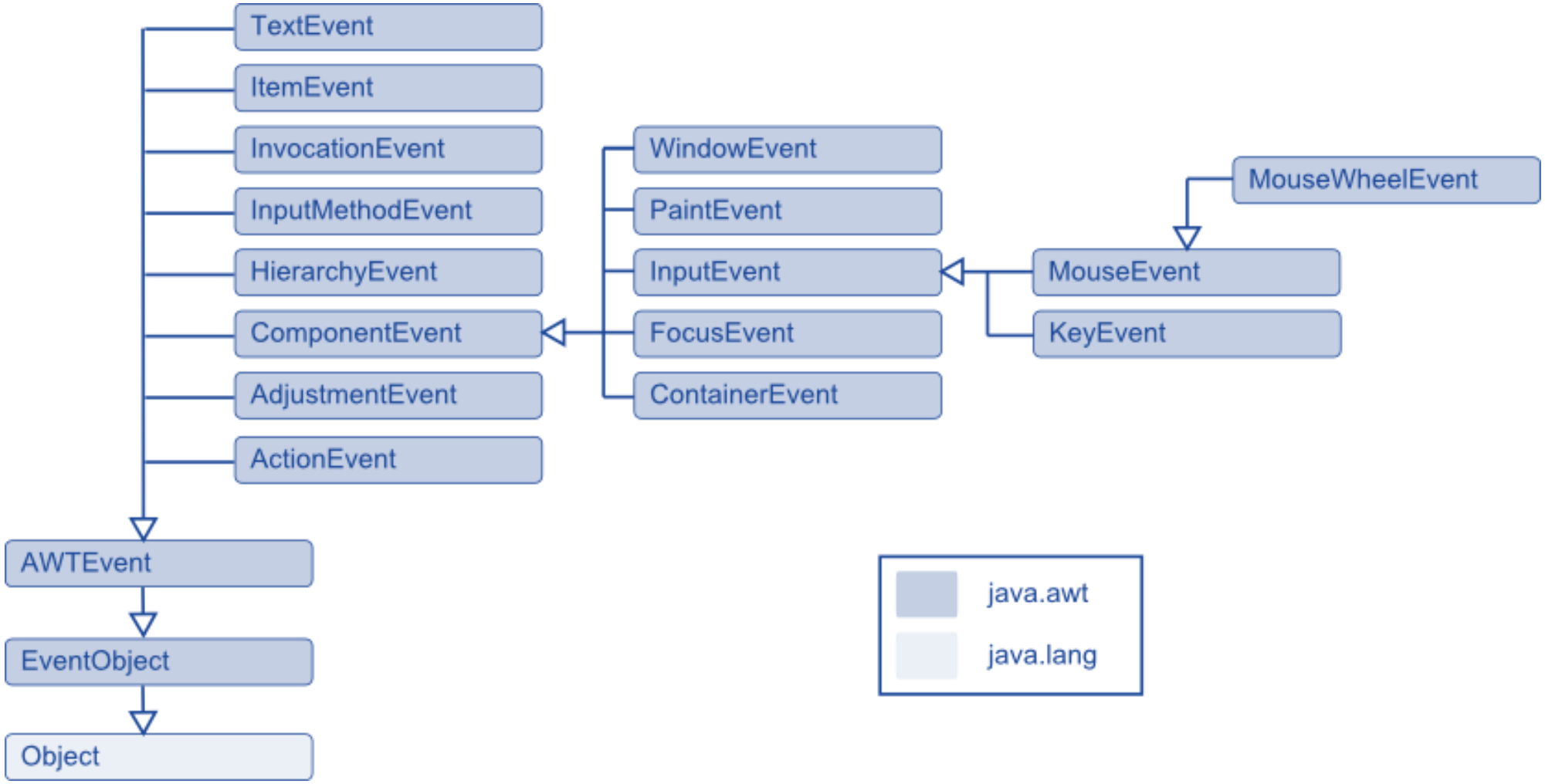
Eventos

- A alteração no estado de um objeto é conhecida como Evento, ou seja, o evento descreve a mudança no estado da origem

- Os eventos são gerados como resultado da interação do utilizador com os componentes da interface gráfica

- Por exemplo, clicar num botão, mover o rato, digitar um caractere pelo teclado, selecionar um item de uma lista e fazer *scroll* a página são atividades que despoletam um evento

Eventos


TextEvent
ItemEvent
InvocationEvent
InputMethodEvent
HierarchyEvent
ComponentEvent
AdjustmentEvent
ActionEvent
WindowEvent
PaintEvent
InputEvent
FocusEvent
ContainerEvent
MouseWheelEvent
MouseEvent
KeyEvent
AWTEvent
EventObject
Object
java.awt
java.lang

Manipulação de Eventos

- Java utiliza o **Delegation Event Model** para manipular os eventos. Este modelo define o mecanismo padrão para gerar e manipular os eventos com os seguintes participantes:

- **Source** - A origem é o objeto no qual o evento ocorre. É responsável por fornecer informações do evento ocorrido ao seu manipulador

- **Listener** - Também é conhecido como manipulador de eventos. É responsável por gerar uma resposta para um evento. O *listener* também é um objeto. O *listener* aguarda até receber um evento. Quando recebe um evento, processa o mesmo e em seguida, retorna.

Passos envolvidos no tratamento de eventos

- Passo 1 - O utilizador clica no botão e o evento é gerado
- Passo 2 - O objeto da classe de eventos em questão é criado automaticamente e as informações sobre a origem e o evento são preenchidas no mesmo objeto
- Passo 3 - O objeto de evento é encaminhado para o método da classe ouvinte registada
- Passo 4 - O método é executado e retorna

Registar o *listener* para um determinado evento

```
jButtonConvert = new javax.swing.JButton();

jButtonConvert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonConvertActionPerformed(evt);
    }
});
```

Registar o action listener para ficar à escuta de eventos

Instruções executadas quando o evento é despoletado

Novo evento criado

Manipular o evento registado previamente

Método invocado quando o evento é despoletado

```
private void jButtonConvertActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:

}
```

Cabe ao programador escrever as instruções que pretende executar

Evento despoletado é passado como argumento

Registar o *listener* para um determinado evento

```
jButtonConvert = new javax.swing.JButton();

jButtonConvert.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButtonConvertActionPerformed(evt);
    }
});
```

Registar o action listener para ficar à escuta de eventos

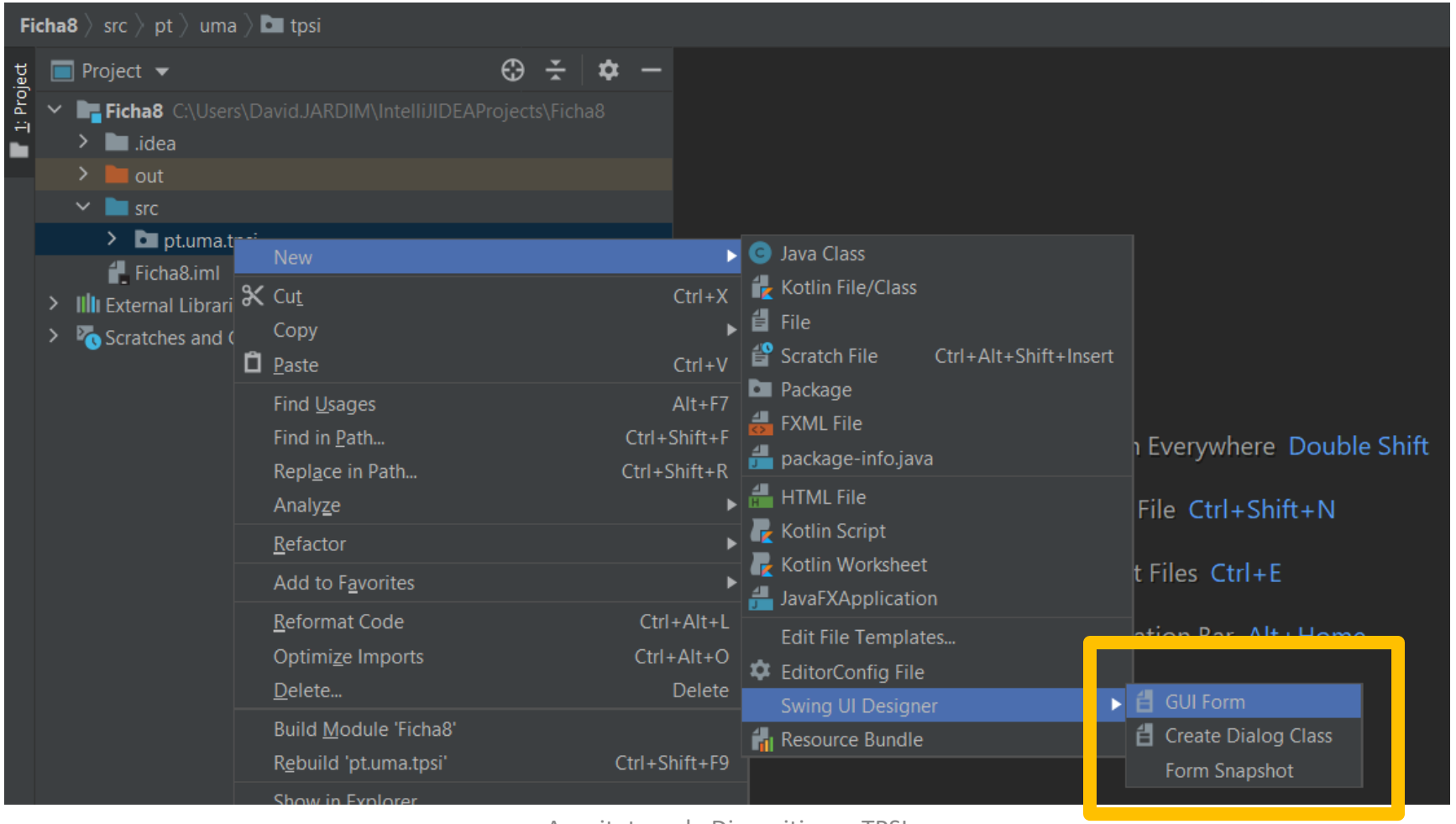
Instruções executadas quando o evento é despoletado

Exemplo


```
okButton.addActionListener(new ButtonClickListener());
submitButton.addActionListener(new ButtonClickListener());
cancelButton.addActionListener(new ButtonClickListener());
```

```
private class ButtonClickListener implements ActionListener{
   public void actionPerformed(ActionEvent e) {
      String command = e.getActionCommand();

      if( command.equals( "OK" ))  {
         statusLabel.setText("Ok Button clicked.");
      } else if( command.equals( "Submit" ) )  {
         statusLabel.setText("Submit Button clicked.");
      } else {
         statusLabel.setText("Cancel Button clicked.");
      }
   }
}
```

Como criar um formulário



Implementação da classe para o formulário

```
public class JavaSwing extends JFrame {
    public JavaSwing(String title){
        super(title);
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.setContentPane(jPanelMain);
        this.pack();
    }

    public static void main(String[] args) {
        JFrame frame = new JavaSwing( title: "Java SWING Examples");
        frame.setVisible(true);
    }
```


UNIVERSIDADE da MADEIRA
CTeSP
CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS